ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Papyrus, EATOP, and MetaEdit+: A comparison between the EAST-ADL modeling tools.

AUTOR: Ana Isabel López Castillo

TITULACIÓN: Grado en Ingeniería Telemática

TUTOR:

- Raluca Marinescu (Doctoral Student)
- Cristina Seceleanu (Associate Professor)

UNIVERSIDAD: Universidad de Mälardalen

CENTRO: Escuela de Innovación, Diseño e Ingeniería.

PAÍS: Suecia

Fecha de lectura: 2015/06/16

Calificación:

El Coordinador de Movilidad,

Resumen

Diversos Lenguajes de Descripción de Arquitecturas (ADLs) están surgiendo como modelos para describir y representar arquitecturas de sistemas. Entre ellos es destacado el lenguaje EAST-ADL, que representa una abstracción de los sistemas de software embebido para automóviles. Ante la necesidad de implementar el lenguaje EAST-ADL, han surgido diversas herramientas de modelado que llevan a cabo esta tarea. El alcance de este proyecto consiste en una comparación detallada de tres editores EAST-ADL: Papyrus, EATOP y MetaEdit+, proporcionando un marco conceptual, describiendo los criterios de comparación y finalmente ejemplificando con el caso de uso Brake-By-Wire que nos ha sido proporcionado, y cuyo desarrollo no es sujeto de este proyecto. La motivación para desarrollar este proyecto parte de proporcionar al usuario una guía comparativa de estas tres herramientas de modelado para facilitar su elección a la hora de desarrollar su trabajo.

Abstract

Several Architecture Description Languages (ADLs) are emerging as models to describe and represent system architectures. Among others, EAST-ADL language is highlighted. It represents an abstraction of embedded software systems for automobiles. Given the need to implement the EAST-ADL language, there are many modeling tools to perform this task. The scope of this thesis is a detailed comparison of three EAST-ADL editors: Papyrus, EATOP and MetaEdit +, providing a conceptual framework, describing the comparison criteria, and finally exemplifying thanks to the Brake-By-Wire use case which has been provided, and whose development is not the subject of this project. The motivation for developing this project is to provide comparison guide between these three modeling tools to facilitate developers choice when deciding the tool in which develop their work.

Table of contents

1. Introduction

Among the new improvements in motor systems that have appeared in recent decades, architecture description languages (ADLs) are included. Automobile companies have a special interest in these modeling languages as they are viewed as a solution key to improve the service quality of automotive electronic and software systems. Vehicles have been composed of mechanical and hydraulic systems to satisfy the needs so far; however, through the emergence of ADLs, electrical/electronic systems have replaced former systems meeting the same requisites successfully, and adding more complex systems capabilities that provide new vehicle features. Between the different existing ADLs, this thesis will be based on EAST-ADL [1]. This architecture description language provides an approach to the description of automotive electronic systems through a model that structures information captured [1]. EAST-ADL has as challenges to increase the functionality, complexity, safety and quality of automobile systems. In order to achieve the challenges, EAST-ADL proposes a system modeling approach framework that organizes and represents captured information, provides separation of concerns through abstraction levels, and remains linked to AUTOSAR specification which makes a considerable contribution at implementation level. Since EAST-ADL has been a subject to implement, several modeling tools have been developed to achieve this task. Modeling tools provide EAST-ADL editing functionalities, frequently graphical and expandable to requirements management, model validation, etc. According to modeling tools availability, they can be divided into open-source tools, such as Papyrus and EATOP; and commercial tools such as MetaEdit+. Modeling tools also support interoperability as they have serialization capabilities to import and export information through EAXML files. Some modeling tools also provide optimization capabilities or/and analysis functionalities to verify activities [2]. These and many other features of the modeling tools will be taken into account when comparing Papyrus, EATOP and MetaEdit+ tools, all of them complying with the standard set by EAST-ADL. Furthermore, the Brake-By-Wire use case, provided for this work, will be shown to exemplify tool features and operation.

2. Preliminaries

This section provides a basis for understanding the Architecture Description Language to be used, EAST-ADL, and three tools that implement it and will be the subject of comparison: Papyrus, EATOP and MetaEdit+.

2.1 EAST-ADL

EAST-ADL is an Architecture Description Language that captures engineering information of automotive Electrical/Electronic (E/E) systems to provide an integrated system model. This process requires obtaining the information with a high level of detail in order to model, design, analyze and synthesize it. EAST-ADL describes different types of information including vehicle features, functions, requirements, variability, communication, deployment of software and hardware resources, performance constraints such as behavior and timing constraints, dependability and Verification and Validation (V&V) related information. The description of the system is structured in several levels of abstraction, from top level counting the functional objectives of the system, to the lowest level performing communication tasks.

EAST-ADL defines how information can be captured, but does not describe the procedure of analysis and synthesis thereof. This description language, therefore, defines a set of design artifacts that may be developed using company specific processes, in order to allow information exchange between tools and organizations.

The system model is made up of four abstraction levels, each one has a specific role and gives a different detail level of modeling and a different view of systems, features and functions, finding an independence between levels to avoid concerns.

The four abstraction levels are shown in *Figure 1*.

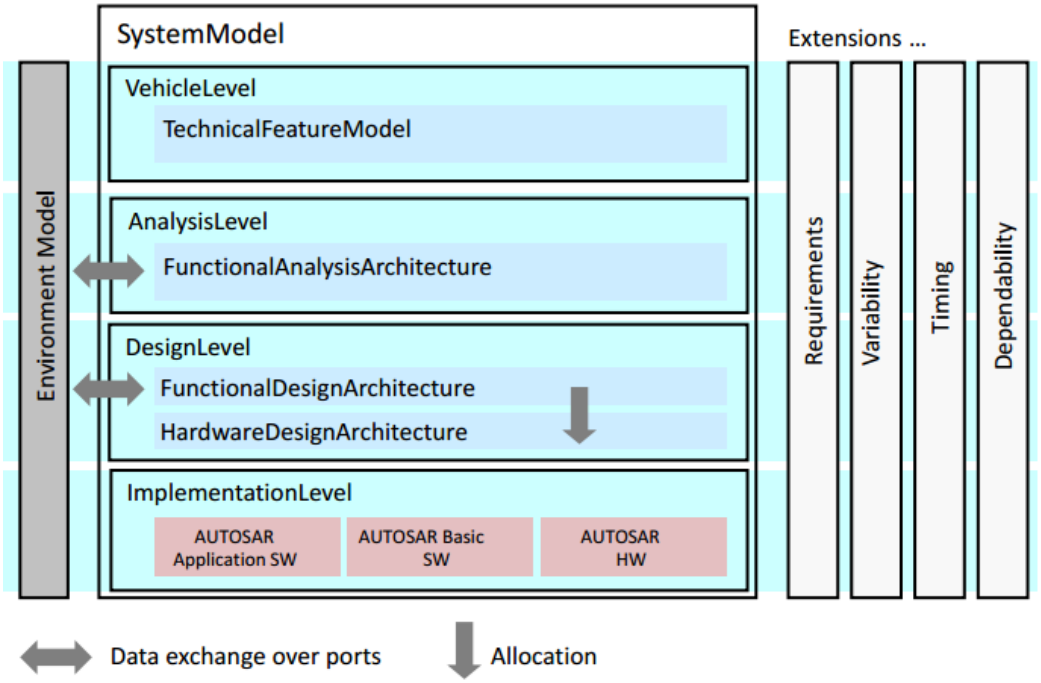

Figure 1: EAST-ADL abstraction levels [3]

1. Vehicle level: describes a basic vehicle feature model and its functional objectives.
2. Analysis level: shows the architecture structure and behavior at a high level, without giving design details or technology implementations.
3. Design level: defines separately the software and the hardware of the system.
4. Implementation level: defines the system implementation in software.

EAST-ADL includes several extensions to the system model, such as requirements, variability, timing, behavior and generic constraints. These extensions reference core elements in all abstractions levels. The information of the extensions is managed in external packages; therefore, the meta-model and users models are modular.

EAST-ADL is also composed of an environment model that captures the behavior of all relevant elements interacting with the E/E architecture.

All aspects of the automotive E/E system are specified in a domain level, providing an efficient management of the relations between these aspects. The EAST-ADL meta-model is adapted as a specification of the domain model and uses concepts from AUTOSAR meta-model, i.e. the EAST-ADL meta-model can be imported into the AUTOSAR meta-model [4]. Additionally to the domain model, the EAST-ADL language is also implemented as a UML2 profile which is used in UML2 tools for user modeling.

The separate representation by models prevents from possible system global problems with the positive contribution of communication between the different levels. In addition, the capability to use product line engineering and with this, the possibility to decompose the structure into components has enable components reusability, increasingly important in the automotive industry.

2.1.1 System model

The EAST-ADL structure is defined dealing with a System Model. The System Model organizes the different models according to each abstraction level and manages the relationships between them.

2.1.1.1 Vehicle level

The Technical Feature Model organizes the core model exposing all the features of the complete system of vehicle level. This feature model may contain what the system provides and which options or dependencies are available in this organization.

2.1.1.2 Analysis level

The analysis level shows the architecture structure and behavior at a high level, without design details or technology implementations. The Functional Analysis Architecture (FAA) contains analysis functions that inspect features in the Technical Feature Model. These functions can be hierarchically composed and connected to each other. Sensors and actuators are directly connected to the environment and represented in the analysis level by functional devices [1].

There are two types of ports hosted in these functions:

- FunctionFlow ports: available for data exchange.

- ClientServer ports: available for client-server communication by means of parameters holding argument or return values [1].

The FAA allows V&V of the vehicle subsystems at a high level of abstraction.

2.1.1.3 Design level

The design level defines separately the software and the hardware of the system. In the Functional Design Architecture (FDA) implemented-oriented aspects are introduced in order to allow a subsequent software decomposition of the functional architecture. In addition, functions from the function structure can be realized by AUTOSAR software components (SW-C).

Design functions and Local Device Managers (LDM) represent application software in the FDA. At this level, there are two types of functions:

- Basic Software Functions (BSWFcn), which model the middleware functionality abstraction in the implementation level.
- Hardware functions (HWFcn), which model the logical hardware [4].

The Hardware Design Architecture (HDA) defines a set of constraints in order to achieve a joint development of the software and hardware applications through iterations and joint performance.

2.1.1.4 Implementation level

The implementation level defines the system implementation in software which relies on the use of AUTOSAR entities that are part of the system model to support traceability.

2.1.2 Environment model

The environmental model captures the behavior of the environment. It also contains functions representing, among other things, other vehicles or road-side IT systems [1].

2.1.3 Extensions

2.1.3.1 Requirements modeling

A requirement is a necessary condition or capability that a system or system's component must satisfy in order to comply with a contract, standard, specification or other properties. There are different types of requirements depending on the level of detail. EAST-ADL group and organizes hierarchically requirements using the Requirement Container construct. The relations between the different requirements are managed by Derive Requirement [4].

Requirements are also classified depending on whether they consider the functionality of the system (functional requirements), or focuses on the non-functional property of the system (quality requirements) [1].

2.1.3.2 Variability modeling

Variability is the property of a system's variant of being changeable by another one in the complete system. This extension involves several levels of abstraction. On the one hand in vehicle level, the Technical Feature Model has been given the responsibility of managing

variabilities of the whole system, providing an overview of these variabilities and the relationships between them through the use of Product Feature Models. On the other hand, these variabilities will be defined and validated in Analysis and Design levels, this is so because EAST-ADL allows the possibility of using feature models also at these levels [1].

2.1.3.3 Timing modeling

EAST-ADL provides timing modeling support on the functional abstraction levels through the Timing Augmented Description Language (TADL). Timing information can be distributed as timing requirements and timing properties, where the latter must satisfy the suitable timing requirement. Furthermore, AUTOSAR has its own timing extensions which model timing requirements and properties in the implementation level.

The timing model extension contains timing constraints which are referenced from the system model. In EAST-ADL the Requirements Modeling Support keeps under view the solutions and verifications from the several requirements.

Timing constraints are defined with the help of Events and Event Chains. Events may be used use in every abstraction level and they are observable due they cause reactions or results. The Timing model imposes timing constraints on event chains. In addition, it is necessary to establish synchronism of events taking into account the number of events that can perform simultaneously [1].

2.1.3.4 Dependability and error modeling

Dependability of a system is measurable by the ability of ensure that service failures do not exceed a maximum threshold of frequency and severity. It covers many aspects such as reliability, availability, safety, maintainability and integrity [4].

Through the representation of inappropriate or abnormal behaviors of the system, the Error Modeling complements the development of security protocols. It also captures information about the failure behavior providing as a result a list of system failures [1].

2.1.3.5 Behavior Constraint Modeling

EAST-ADL gives the user the capability to make precise and integrated annotations of different parts of the project such as requirements, implementation, application modes and functions, anomalies and resource deployment. The language features give the user the opportunity of tracing and maintaining the engineering concerns related and the analytical information gathered using EAST-ADL.

EAST-ADL implements a hybrid-system semantic with capability to support behavioral annotation concerns. Due to the formal semantics of EAST-ADL and the behavior constraints applied to the language that can be associated in terms of time, a user can define a explicit model transformation from EAST-ADL to another model format of other analysis method or tool [1].

2.2 Papyrus

2.2.1 Introduction

Papyrus is composed by a UML and a SysML graphical editor which work under the Eclipse Model Development Tools project.

2.2.2 Background

Nowadays, the Unified Modeling Language (UML) is used all over the world in a broad variety of projects, either industrial or research ones. This also means that it is taught in a lot of schools and universities offering computer sciences studies. In the last years, at system engineering field, SysML has appeared as a symmetrical effort to have a unified modeling language for this field. These two languages together gather a huge user community all around the world [5].

The fact that Eclipse is a platform under open source licenses allows the UML tool to be used in industry, research projects or proprietary products, with unimpeded access to internal components to modify or extend [5].

2.2.3 UML2 graphical modeler

The main goal of Papyrus is to provide a UML2 graphical modeler based on Eclipse environment. Apart from modeling, Papyrus is able to generate code in various languages such as C, C++ and Java. It facilitates the connection between external tools in order to allow models to be the starting point of development.

UML2 graphical editor is provided by Eclipse and its use is one of the key objectives of the Papyrus development. Papyrus supported diagrams are: use case diagram, class diagram, component diagram, sequence diagram, activity diagram, state machine diagram, composite structure diagram and deployment diagram [6].

2.2.4 Customizability

Eclipse provides much freedom to Papyrus, amongst other things, when it comes to customize profiles. Profiles allow the adaptation of the language to model specific aspects. This UML feature is also very useful to adapt the language to processes and business domains. In addition, there is the option to customize the set of tools by static profiles, besides applying validation rules. The stereotypes are additional modeling concepts that might be associated with specific rules [6].

A user can customize the graphical representation of some of the model's elements depending, for instance, on the type of the model to be represented or even the value of one of the elements defined inside the model. This customization replaces the UML default icons with other ones much more appropriate that allow a more accurate representation of the model [6].

2.2.5 Scope

Papyrus provides the perfect combination of SysML and UML diagram editors and other MDE tools, as well as support for profiling mechanisms in these tools.

Given a set of diagrams determined editors, the aim will be to provide the best possible form of integration for each type of diagram. However, sometimes the development of specific diagram editors will be needed as existing do not meet the needs.

On the one hand, Papyrus' support of UML and SysML and its profiling mechanisms, together with the support for collaborative work and the ability to be included in qualified industrial processes, has become the first approach to obtain completeness. On the other hand, editing and extending diagrams properties along with the ability to customize the graphical interface defined in the designs of UML and SysML profile, provide an open and flexible approach.

Summarizing Papyrus objective is to provide the means for integration and extension diagrams editors of SysML and UML tools, as well as profiles design, modeling tools and support mechanisms [5].

2.3 EATOP

2.3.1 Introduction

EATOP (abbreviation of “EAST-ADL Tool Platform”) is an open source project based on Eclipse as an implementation of the EAST-ADL standard.

2.3.2 Scope

One of the main goals of EATOP is the Eclipse Modeling Framework (EMF) implementation of EAST-ADL meta-model. In addition, EATOP has the purpose of serialize models or files in order to comply with that is established by EAST-ADL XSD Schema. Another main feature of EATOP is to provide the platform for users to create, edit, validate and process EAST-ADL models in Eclipse.

There have been efforts to create several implementations of EAST-ADL based on Eclipse so far. However, eventually resulting approximations lacked order or were redundant. Therefore, EATOP was created to reconcile all these initiatives and consolidate disparate implementations in order to create the reference implementation of EAST-ADL into the Eclipse platform.

EAST-ADL and EATOP maintain a close relationship that complements and causes that EATOP is closely aligned with Artop. To allow high interoperability between EAST-ADL and AUTOSAR tools, a platform called Sphinx and its modeling tools will be the base of EATOP. Additionally this will enable the creation of integrated tool chains that provide a proper connection between EAST-ADL and AUTOSAR for transmissions [7].

2.3.3 EATOP capabilities

EATOP has several capabilities define below:

- Thanks to EAST-ADL meta-model is based on EMF, there is a set of tools that generate XML Schema and Ecore meta-model, useful for EAST-ADL releases, as well as a set of design tools that use Java APIs in order to manipulate EAST-ADL model instances. In addition, EATOP contains the meta-model implementation of each EAST-ADL release.

- In respect of management persistency of database and files, EATOP in combination with Sphinix performs serialization/deserialization of the instances of the EAST-ADL meta-model within an EAXML file. Furthermore, EAST-ADL models management, with its database persistence, are located in one or more XML files in the Eclipse workspace.
- Eclipse provides a basic user interface that supports, among other purposes, a wizard for creation of an EAST-ADL project and another one for the creation of an EAST-ADL file. Furthermore, the basic user interface supports an EAST-ADL release preference page, a property page, a property tab and an EATOP perspective and explorer view.
- Interoperability capability and connectivity with other tools and platforms are needed in order to provide a proper development process [7].

2.4 MetaEdit+

2.4.1 Introduction

MetaEdit+ is a multi-tool environment for meta-models development that allows users to create, edit, manipulate and represent, with EAST-ADL language, different architecture models. Furthermore, MetaEdit+ enables static analysis, document generation, generation of other models and code from the EAST-ADL models [8].

2.4.2 Multi-project

MetaEdit+ technology enables the possibility of working in several projects at the same time, even projects that use different modeling languages. Each project is composed of multiple models and these projects can share and reuse data. Furthermore, when a set of component composes a repository, the content of each project is not need to be similar.

2.4.3 Modeling languages

MetaEdit+ supports a great variety of different modeling languages. MetaEdit+ tool uses a selection of predefined modeling languages instead of developing its own language. The number of available modeling languages supported in MetaEdit+ is quite high due to it is based on meta-models. The use of these languages is usually for reference or example purposes but they can also be used for real-life production purpose. Some of these modeling languages are the following ones: UML, EAST-ADL, GOPRR and CPL [9].

2.4.4 The Object Repository

MetaEdit+ can work as a single-user workstation environment, or as a multi-user environment in which many workstation clients are connected by a network to a server. In both cases, there is an Object Repository which contains all the information related to the different models and modeling languages. This information stored in the repository must be updated if the system design or the modeling language is modified, since this is the basis for the generation of code and documentation, and also in multi-user environment, changes made from a client must be visible for other clients. Locks are used to prevent users modify data at the same time [9].

3. Comparison criteria

In order to provide a proper comparison between the different modeling tools, the following proposal has been meditated and considered in next section.

3.1 Tooling aspects

On the one hand, it has been decided to group the set of features related to the tool:

- **Availability**: refers to the accessibility to the tool, in terms of the process of obtaining it.
- **Usability:** includes three aspects that define the usage of the tool.
  a) Documentation: describes the amount of information provided from the tool's platforms.
  b) Installation: refers to the installation process, including installation prerequisites, characteristics, troubles, etc.
  c) User interface: describes the tool interface, distinguishing between non-graphical and graphical interface that allows users to interact with the models through a graphical representation of them.
- **Interoperability:** refers to the capability of the tool to share data between different systems or tools.
  - Output file: describes the files regarding to models.

3.2 Modeling aspects

On the other hand, it has been decided to group the set of features related to the modeling development:

- **Structural information**
  a) System level modeling: describes the system through its levels of abstraction, focusing on function types, function prototypes and connectors.
  b) Component level modeling: describes the component interface focusing on elements such as ports and triggering functions.
- **Behavioral information**
  a) Functional behavior modeling: describes the capacity of the tool to provide a functional behavior to the models.
  b) Functional extra-behavior modeling: describes the possibilities of adding extra-information such as timing and generic constraints.

4. Application of the comparison criteria on the tools

This section applies the comparison criteria between three modeling tools (Papyrus, EATOP and MetaEdit+) defined before and follows the structure of section 3.

4.1 Tooling aspects

4.1.1 Availability

Papyrus is an open-source UML modeler which can be downloaded without charge from Papyrus Eclipse website [10].

EATOP is also an open-source editor obtainable from EATOP Eclipse website [11].

MetaEdit+ tool is a commercial tool from MetaCase. Users have two options:

- Download a free 31-day evaluation version of full MetaEdit+ Workbench.
- Purchase for different kits with licenses and services.

Both options are available in MetaCase webpage [12].

4.1.2 Usability

a) Documentation

Papyrus provides some tutorials and examples showing customizing capabilities. Moreover, it provides "Papyrus Wiki", where users can find more information, and "Papyrus forum", in which users can ask questions and discuss with other users. All this documentation is easily accessible as it is located on Papyrus official website [13]**.**

EATOP provides little information at Eclipse EATOP proposal webpage [7], not as large as Papyrus or MetaEdit+, which entails an obstacle to get an affordable learning. Some tutorials, wiki pages and discussion sections are available in order to provide basic concepts to users. Other tabs available are a view source and a history tab [14].

MetaEdit+ provides a configuration guide, tutorials [8], updates and exercises that enable users to become familiar with the tool. As Papyrus and EATOP, MetaEdit+ also has available forums and FAQs sections. Users can easily access the MetaEdit+ manual [15] in order to find more information.

More documentation of these three tools is provided by MAENAD [16].

b) Installation

Papyrus requires the installation of the Eclipse Modeling Platform [17], and then the installation of Papyrus modeling component [18]. Updates of this platform can be obtained easily thanks to Eclipse software updates. Eclipse provides several versions that support Papyrus (Eclipse Luna/Kepler/Juno/etc.)

EATOP uses also the basic Eclipse user interface. However, EATOP requires the installation of a Java Runtime Environment by means of Java Development Kit (JDK). In this case, to provide a proper analysis comparison is enough to make use of the EATOP demonstrator [19] which is the simplest way of getting the modeling interface

comparing to the other two tools, since it is only required to download the folder and execute the demostrator.

MetaEdit+ requires the tool installation [12]. After that, an extraction of the EAST-ADL repository file is needed in order to place this file into the same folder where MetaEdit+ has been installed.

c) User interface

Papyrus and MetaEdit+ tools provide a graphical representation of EAST-ADL models. However, this feature is not available in EATOP tool.

Papyrus provides several views including a main view, which shows the multiple model diagrams that can be arranged in tabbed views and its items are selectable. Other views are: an outline view, a property view and a bird's-eye view. Furthermore, all these views can be created, placed or re-sized as user desires, providing to this tool a highly customizable feature. Papyrus provides another property that is to add new types of diagrams from different technologies compatibles with Eclipse through a diagram plug-in mechanism.

EATOP demonstrator is based on Eclipse graphical interface. It provides a set of editors for modeling the EAST-ADL levels of abstraction. There is a root diagram which contains all the elements of the EAST-ADL model. Developers, through the editors, can access to each model in the disaggregation of the main package.

MetaEdit+ graphical tool has a nimble behavior, allowing the use of libraries available or generating its own plots (with basic tools) and even icons. Code generation is very versatile allowing any transformation. This way, results are achieved quickly and obtaining a clear definition of model and meta-model. MetaEdit+ allows users to design and modify graphical representations through the use of browsers and editors such as graphical, matrix or table editors. The Symbol editor is used by developers with meta-modeling rights to design its models.

4.1.3 Interoperability

Papyrus provides support for UML and SysML modeling. The tool through the UML2 implementation complies with OMG standard. The OMG Diagram Interchange manages models graphical interoperability connecting different tools. Other OMG standards supported by Papyrus are MARTE, CCM and LwCCM. Papyrus enables analysis tools such as Hip-hops and Qompass. On the other hand, Papyrus has supported the development of several editors through different technologies such as EMF Tree Editors, GMF, GEF and SWT.

EATOP is based on the Sphinx modeling tool platform which provides interoperability between EAST-ADL and AUTOSAR. The model/file serialization and thus the creation of EAXML support interoperability. In addition, there are some platforms such as CESAR or MBAT with domain independent abstractions of EAST-ADL that provide interoperability. EATOP has a close relationship with ARTOP [20] with whom it shares common features and functionalities to create modeling tools under the support of AUTOSAR.

MetaEdit+ supports a wide variety of interoperability approaches. First of all, it supports an API which enables users to design data and functions from their own applications. The definitions of API commands for these applications are contained in a WDSL file. Furthermore, the API interface is implemented as a SOAP Web Service server. Both, SOAP server and the WSDL file are provided by MetaEdit+ in order to support calls via SOAP. In addition, a set of command line parameters is carried by MetaEdit+ in order to support the operations for patches representations and the configuration of the optional second monitor.

Another interoperability feature is the generators access to the repository in order to transform the information stored into text-based outputs. MetaEdit+ offers generators to check or review the design results and perform the generation of code and documentation. Finally, it is possible to import and export XML format files in the repository. These XML files contain graph information including objects and properties [6].

- Output file

  Papyrus. The creation of a UML model in Papyrus involves the generation of three files:

  - ****.di***: contains the tool metadata and constitutes the entry point for starting modeling.
  - ****.notation***: contains graphical data.
  - ****.uml***: contains UML model data.

  All these files use XML Schema language providing a hierarchical structure of the contents. Papyrus enables to export model information in EAXML file.

  EATOP. The content of EAXML files conformed to the XML Schema obtained as result of the serialization/deserialization process. It contains the serialized instances of the EAST-ADL meta-model.

  MetaEdit+. It is possible to generate outputs of the information stored in diagrams. Thanks to the generation system, model information can be exported in Word form generation or HTML form. MetaEdit+ also allows exporting the package that contains all diagrams hierarchically in an EAXML file.

4.2 Modeling aspects

4.2.1 Structural information

In order to obtain appropriate and not overly complex results, the comparison between the three tools will be based on Design level.

a) System level modeling

Papyrus provides a graphical EAST-ADL structure where users can interact with the different abstraction levels, selecting them to create UML class diagrams. For instance, selecting the Design level, users can create a class diagram which represents the Functional Design Architecture (FDA), the Hardware Design Architecture (HDA) and the Allocation. Making use of the Model explorer view, users can create:

1. A class diagram for defining DesignFunctionTypes.
2. A composite structure to represent the FDA and add DesignFunctionPrototypes to it through the use of the palette.
3. A composite structure to represent the HDA and add HardwareComponentPrototypes to it through the use of the palette.
4. An Allocation table and add different AllocationFunctions to it. The process to create a FunctionAllocation consists of two steps:
    - Define *Target* in the table using Properties view.
    - Define *Allocated element* using Properties view.

Properties view, is also used for defining types. This way, users can select a certain prototype and define its type from the available ones.

Papyrus also allows creating different tabs in a model. This way, developers can group in the same model FDA FunctionTypes diagram, FDA diagram, HDA FunctionTypes diagram, HDA diagram and Allocations diagram. All these diagrams will represent Design level.

Connectors are represented in the HDA diagram, showing a set of variables that give information about pins connected and path.

As mentioned above, Papyrus provides a Palette in order to facilitate for developers the fact of adding EAST-ADL levels, functions, hardware components, etc. Some of the palette possibilities are shown in *Figure 2*.

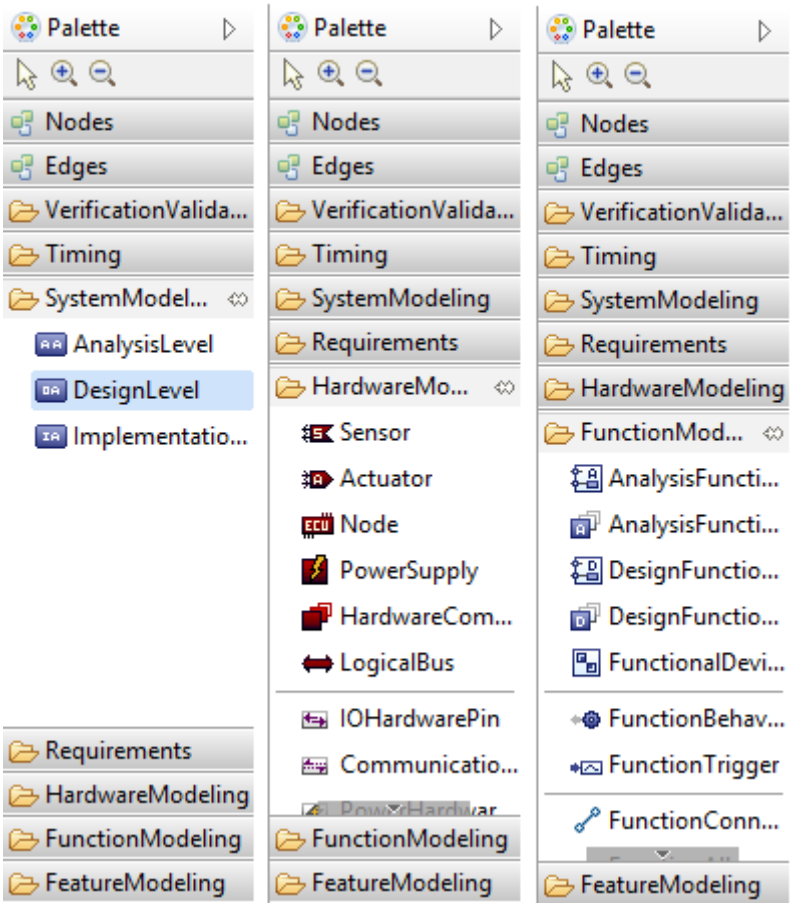

Figure 2: Palette from Papyrus

Allocation diagram shows the allocations between HDA FunctionPrototypes and FDA FunctionPrototypes. Furthermore, it also shows the corresponding function allocation by selecting a particular component, and the FunctionAllocation is shown as a dependency with an "allocation" keyword attached to it.

EATOP follows a similar process as Papyrus in terms of EAST-ADL elements creation. However the main view in EATOP is the EAST-ADL explorer view. It allows users to browse the model from the tree. This way, an eaxml file is needed in order to store the model. Pending from the EAXML element there is a hierarchical display of the model's content.

The first Explorer level consists of EAPackages. One of them is TopPackage which contains VehicleLevel, AnalysisLevel and DesignLevel. Focusing on DesignLevel, three elements pend from it: FDA, HDA and Allocation. The DesignLevelElements EAPackage is also noteworthy, it contains two subpackages with DesignFunctionTypes and HardwareComponentTypes respectively. Thereby, DesignFunctionPrototypes and HardwareComponentPrototypes located in TopPackage refer to them.

The FDA is represented as a DesignFunctionPrototype defining the type in Properties View. The referred type is located in AnalysisLevelElements EAPackage. The FDA is composed of a list of children that can be identified as DesignFunctionPrototypes and FunctionConnectors. If these children are written in italic means that they come from the type and not from containment. As a feature, EATOP's DesignFunctionPrototypes can be composed of other DesignFunctionPrototypes.

EATOP allows editing a referred DesignFunctionType making use of the properties view. After editing a certain type, by clicking the reference itself is possible to see that the default Eclipse editing is still available and the prototype has been filled in automatically.

The HDA HardwareComponentPrototype has a similar structure as FDA, despite the fact that HDA is composed of HardwareComponentPrototypes and HardwareConnectors.

The Allocation element contains AllocationFunctions in which the relations between the allocateable elements and the allocation targets are shown through the instances references submenu.

EATOP use icons to recognize easier the different elements. Some of the elements of Design level are distinguished in *Table 1*.

	EAPackage		
	DesignLevel		
FDA		**HDA**	
	DesignFunctionPrototype		HardwareComponentPrototype
	FunctionConnector		HardwareConnector
	DesignFunctionType		Actuator
	LocalDeviceManager		Node
	FunctionFlowPort		Sensor
Allocation			IOHardwarePin
	AllocationFunction		CommunicationHardwarePin

Table 1: Icons of Design level

When we compare *Table 1* and *Figure 2* which represents icons in Papyrus, we can notice that both tools use the same icons.

EATOP has some additional features for faster navigation between elements based on their references. References are represented with an arrow on the icon, as shown in *Table 2,* indicating that some elements refer to it. It is possible to see referring items and jump to any of them by clicking right on the icon.

	A sensor is instantiated
	An actuator is instantiated

Table2: Icons referenced

MetaEdit+. Once users have successfully logged and connected to EAST-ADL repository, a graph browser is opened to provide different navigation possibilities through models and model elements. By selecting "Create Graph" in the toolbar, users can create a system model and all the elements it contains.

MetaEdit+ provides four browsers: graph browser, type browser, object browser and metamodel browser. Furthermore, these browsers contain three views:

- Projects, which shows the projects selected in the repository.
- Graphs, which shows all the folders and subfolders contained in the project.
  - As feature, it is possible to set filters in this view using the filter box. For instance, selecting the key **:R** only requirements models will be shown.
- Contents, objects contained in the selected folder in graph's view.
  - Also provides a filter box.

MetaEdit+ represents graphically EAST-ADL system model by means of a set of boxes. Each box has its own representation, as shown in *Table 3*.

	Vehicle level
	Analysis level: Analysis function prototype
	Design level: Design function prototype
	Design level: Hardware component
	Design level: Allocations

Table 3: MetaEdit+ icons for system model

If we compare the icons, we can say that MetaEdit+ use different icons from Papyrus and EATOP.

Focusing on design level, there are three noteworthy folders in graph's view, the ones that contain FDA, HDA and Allocations. Users create a graph for each of these design sublevels and then access them by clicking twice on each folder.

The FDA diagram is composed by a set of FunctionPrototypes. In MetaEdit+, users create the FunctionPrototypes first and then define types for the prototypes and pins. Before a type definition is given, all prototypes look the same: white rectangles with the name on the top. To create a type definition for a prototype, users select the option "Open Subgraph". A subgraph window is open, and developers can either select a type already defined or create a new type. New types can be created mainly as a diagram, table or matrix. When a new FunctionType is created, a set of features such as input/output ports are assigned to it, and therefore, to the FunctionPrototypes that instantiate it.

The HDA diagram is created in a similar way as FDA, with the difference that when a HardwareComponentType is defined, it needs to be specified if is SensorType, ActuatorType, NodeType or ElectricalComponentType.

In addition, both FDA and HDA diagrams allow the addition of relationships between diagram elements thanks to the *Relationship* icon. These relationships can be represented as flow connection, hardware connectors, or realizations and allocation relationships.

Allocations are located in the AllocationMatrix. In the table, DesignFunctions are distributed in rows and Hardware components are distributed in columns. When an AllocationFunction exists between two elements, the square will be marked by the projection of a certain column and the corresponding row. Allocations are also visible in graph's view thanks to the option "Show allocation prototype" in Properties view.

b) Component level modeling

Papyrus. At this level, components represent FunctionTypes and FunctionPrototypes. Developers configure FunctionPrototypes ports using FunctionPorts which can be easily added through the palette. Then, these FunctionFlowPorts and FunctionClientServerPort are defined showing in the "Applied stereotypes" list the supported variables such as port's name, direction (in, out) or data value (in case of being a FunctionFlowPort). A set of variables for component interaction are also conserved and listed in the properties view.

On the other hand, in Papyrus users have the possibility to represent FunctionTriggers using the palette. However, the different triggering conditions associated to a FunctionType, and therefore, applied to a FunctionPrototype of the given type, are not shown in Papyrus diagram. Thanks to advanced option in properties view users can configure a port: if triggering is time-driven the association variable will be empty, if triggering is event-driven the association variable must be not empty.

EATOP. The DesignFunctionPrototypes are composed of ports, which in turn define the data type of FlowFunctionPort.

EATOP allows adding EventFunctions, either ClientServePort or FlowPort thanks to the timing extension from the extension's EAPackage. Time and event constraints are associated to these event functions, which also refers to the triggering functions. As difference from Papyrus, FunctionTriggers are not directly configurable in EATOP.

FunctionConnectors and HardwareConnectors include all the information related to the connection between components referencing function port and function prototype; or in terms of hardware, referencing hardware component prototypes and hardware pin.

MetaEdit+. When pressing the Relationship button, it triggers the creation of any kind of relationship. Developers can specify relationship properties in the opened dialog window. In case of being a relationship established between components of the FDA, the first tab shows the flow connection properties, and the two remaining tabs present the input flow port and the output flow port, respectively. In case of being a relationship established between components of the HDA, the main tab shows hardware connection properties and the other two tabs present the connection pins. MetaEdit+ knows EAST-ADL properties and is able to identify the type of connection depending on the source and the destination, as well as the direction of the connection.

Users can add port connections in their HDA diagram with the creation of an object HardwarePortConnection, specifying among other features, the time if it is either TimeTriggered, EventTriggered or both. Users also must indicate the name of the relationships which will use the HardwarePortConnector.

4.2.2 Behavioral information

a) Functional behavior modeling

Papyrus provides a representation of FunctionBehavior. Developers can create FunctionBehaviors by selecting them in the palette. Then, these FunctionBehaviors are associated to FunctionTypes or, directly, to FunctionPrototypes.

EATOP's EAXML element includes an EAPackage which contains EAST-ADL extension elements. Users are allowed to add a FunctionBehavior in this package, by means of the option "New Child".

MetaEdit+. As shown in *Figure 3*, adding a FunctionBehavior is not an option available in the sidebar.

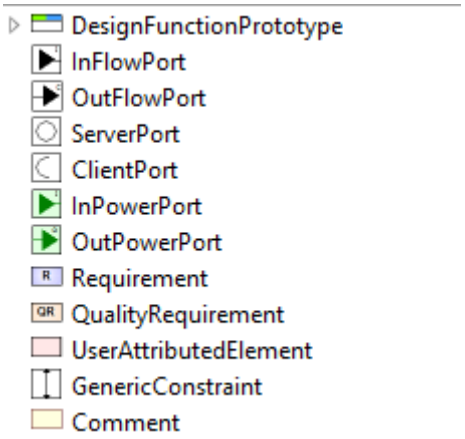

Figure 3: MetaEdit+ sidebar

b) Extra-functional behavior modeling

Papyrus.

- Time constraint:
  Papyrus allows user to add precedence constraints, execution time constraints, input and output synchronization constraints, and periodic event constraints thanks to the palette provided. Each of these constrains has several features related to function types and function prototypes easily configurable.
- Generic constraint:
  Papyrus does not provide generic constraint in the list of possible elements to add to a diagram.

EATOP.

Users have the possibility of adding timing and generic constrains among other conditions in the extension elements EAPackage.

MetaEdit+.

As shown in *Figure 3*, generic constraints are easily added as an option available in the sidebar. However, timing constraints are not available in the sidebar.

4.3 Comparison overview

Table 4 summarizes the comparison shown in previous section.

		Papyrus	EATOP	MetaEdit+
Tooling aspects	Availability	Open-source	Open-source	Commercial tool
	Documentation (Usability)	Tutorials and examples	Little information	Tutorials and exercises
	Installation (Usability)	Eclipse plugin	Eclipse plugin	Tool installation
	User interface (Usability)	Graphical model representation	Lack of graphical model representation	Graphical model representation
	Output file (Interoperability)	Model, uml and notation files	Model pends from EAXML file	Code generation
Modeling aspects	System level (Structure information)	According to EAST-ADL standard	According to EAST-ADL standard	According to EAST-ADL standard
	Component level (Structure information)	According to EAST-ADL standard	Allows for nested types	According to EAST-ADL standard
	Functional behavior (Behavioral information)	According to EAST-ADL standard	According to EAST-ADL standard	Lack of Function Behavior
	Extra-functional behavior (Behavioral information)	Lack of Generic Constraint	According to EAST-ADL standard	Lack of Timing Constraint

Table 4: Comparison overview table

5. The Brake-By-Wire use case

New automotive systems are being developed in order to carry out standard vehicle operations through the use of electronic components instead of the traditional mechanical methods. This set of systems belongs to the X-by-wire technology [21].

An example of this trend is Brake-by-wire technology, which will be described and applied in this section, providing an exemplification of the modeling tools' use.

5.1 Description

In automotive industry, the Brake-by-wire (BBW) technology is the ability to control brakes by electrical means rather than by hydraulic systems [22]. Thus, brake fluids and mechanical parts associated with the brake are eliminated [21].

The operation consists of measuring the force applied by a driver on the brake pedal and then determine, with the help of additional energy recovery information, the amount of brake pressure to be applied through rear brake calipers. The BBW technology can complement an existing brake system or can be designed independently, replacing a set of traditional components such as pumps, hoses or fluids that belong to hydraulic and mechanical control systems, with electronic control systems using sensors and actuators.

In addition, Brake-by-wire systems, also called electro-mechanical brakes (EMB) are environmentally friendly, and also require less maintenance care [21].

Figure 4 shows the general architecture of an EMB system in a drive-by-wire vehicle. The system is mainly composed of the following: sensors, actuators, memory, processors (including an ECU) and communication networks [22].

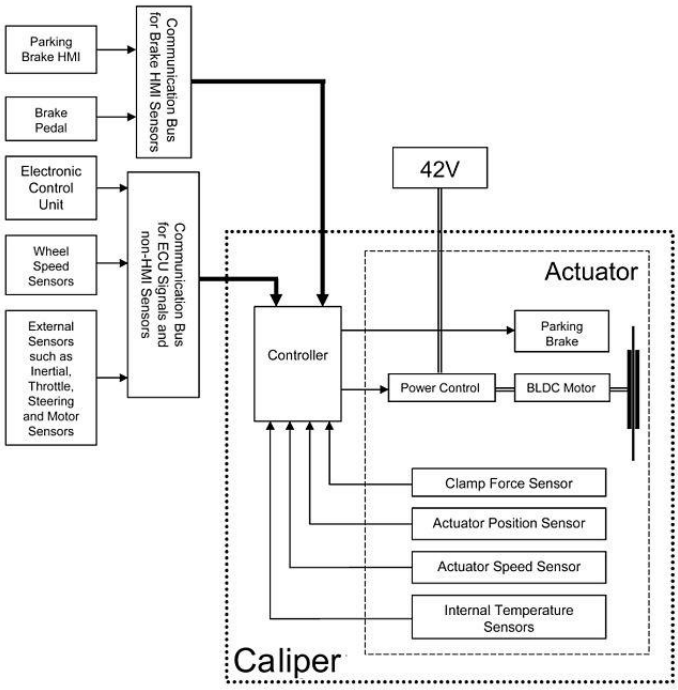

Figure 4: General architecture of an EMB system [22]

5.2 Operation

A vehicle equipped with brake-by-wire technology contains several electronic control units (ECU) connected to the brake pedal receiving information of pedal position and interpreting a set of electronic instructions to then communicate them, through miniatures, to the network that connects the whole braking system in real time. This set of electronic instructions will be forwarded to a small electric motor for each wheel and whose mission is to generate brake pressure [21].

The operation of BBW use case in EAST-ADL involves several components which have its own functionalities and comprises the following steps:

1. The BrakePedalSensor detects pressure on the brake pedal, the WheelSpeedSensor measures the angular velocity of the vehicle and the VehicleSpeedSensor measures the vehicle speed.
2. The torque produced in the pedal brake is measured by the BrakeCalculator and the result of this measure is dent to the BrakeController.
3. The torque distribution of each wheel is calculated by the BrakeController component and the value calculated is sent afterwards to the ABS of every wheel.
4. With the inputs from the BrakeController, the ABS is able to calculate the actual torque to be transferred to each wheel and sends an activation signal to the BrakeActuator.
5. Finally, the BrakeActuator acts as main responsible for the actuation of the physical brakes [23].

This process involves, among other conditions, timing constraints.

Next sections will show the EAST-ADL implementation of the BBW use case in each tool focusing on Design level, which has been compared in section 4. These implementations have been provided in order to facilitate the comparison. For that reason, the implementation development will not be covered in this thesis.

At Design level, Functional Design Architecture, Hardware Design Architecture and Allocations will be taken into consideration.

5.3 BBW in Papyrus

- Functional Design Architecture

  Figure 5 shows FDA diagram in Papyrus. This representation takes into account the pedal brake and the four wheels inputs, with its respective elements such as sensors, encoders, etc. The controller and the speed estimator are also represented. And finally, it shows the wheel's outputs, through its respective elements (ABS, actuators, etc.).

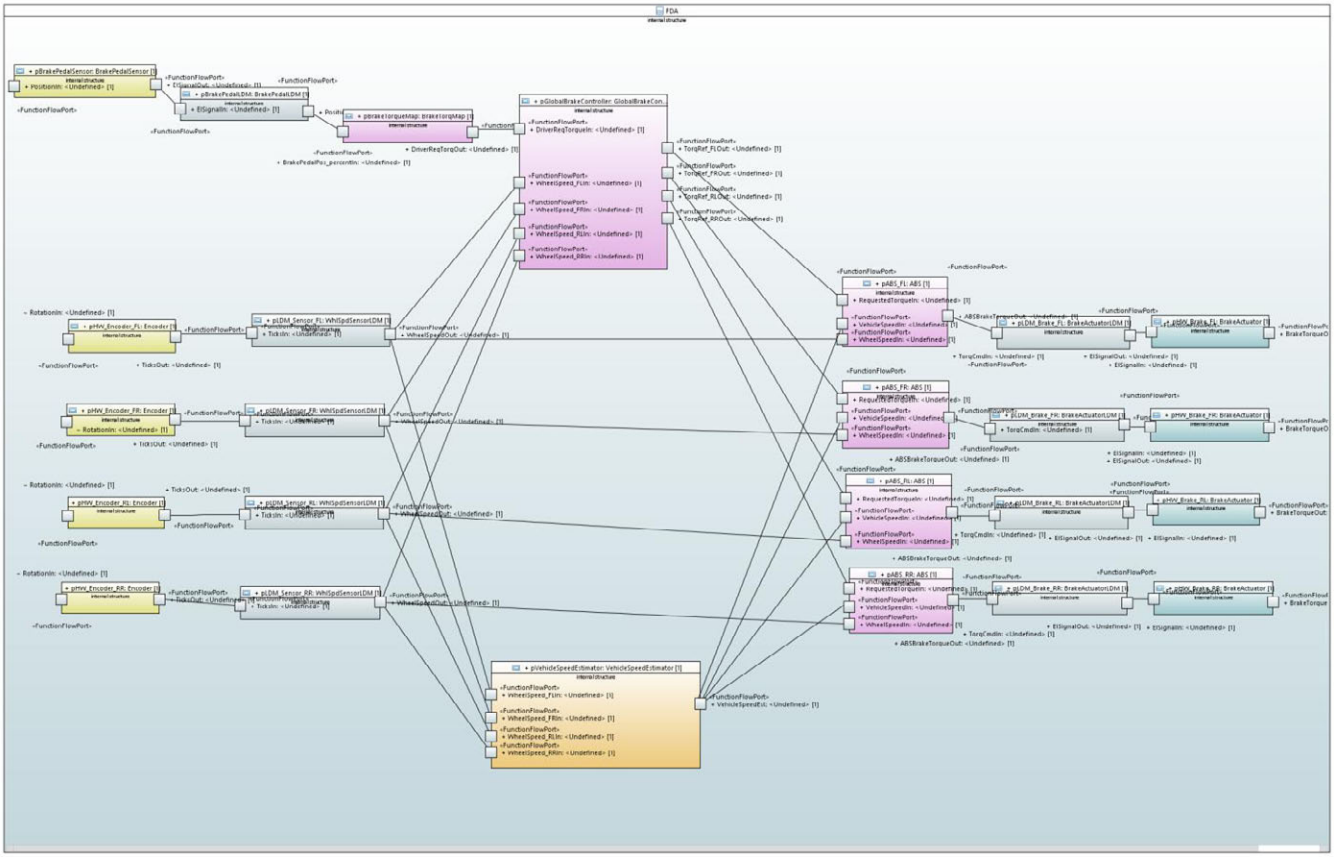

Figure 5: FDA in Papyrus

- Hardware Design Architecture.

  Figure 6 shows HDA diagram in Papyrus. As we can see, there is an ECU for each wheel and another one which is central. Each wheel's ECU receives information from its wheel's encoder and brake actuator. All the ECUs are connected to each other sending its information to central ECU, which sends the pedal position to the pedal sensor.

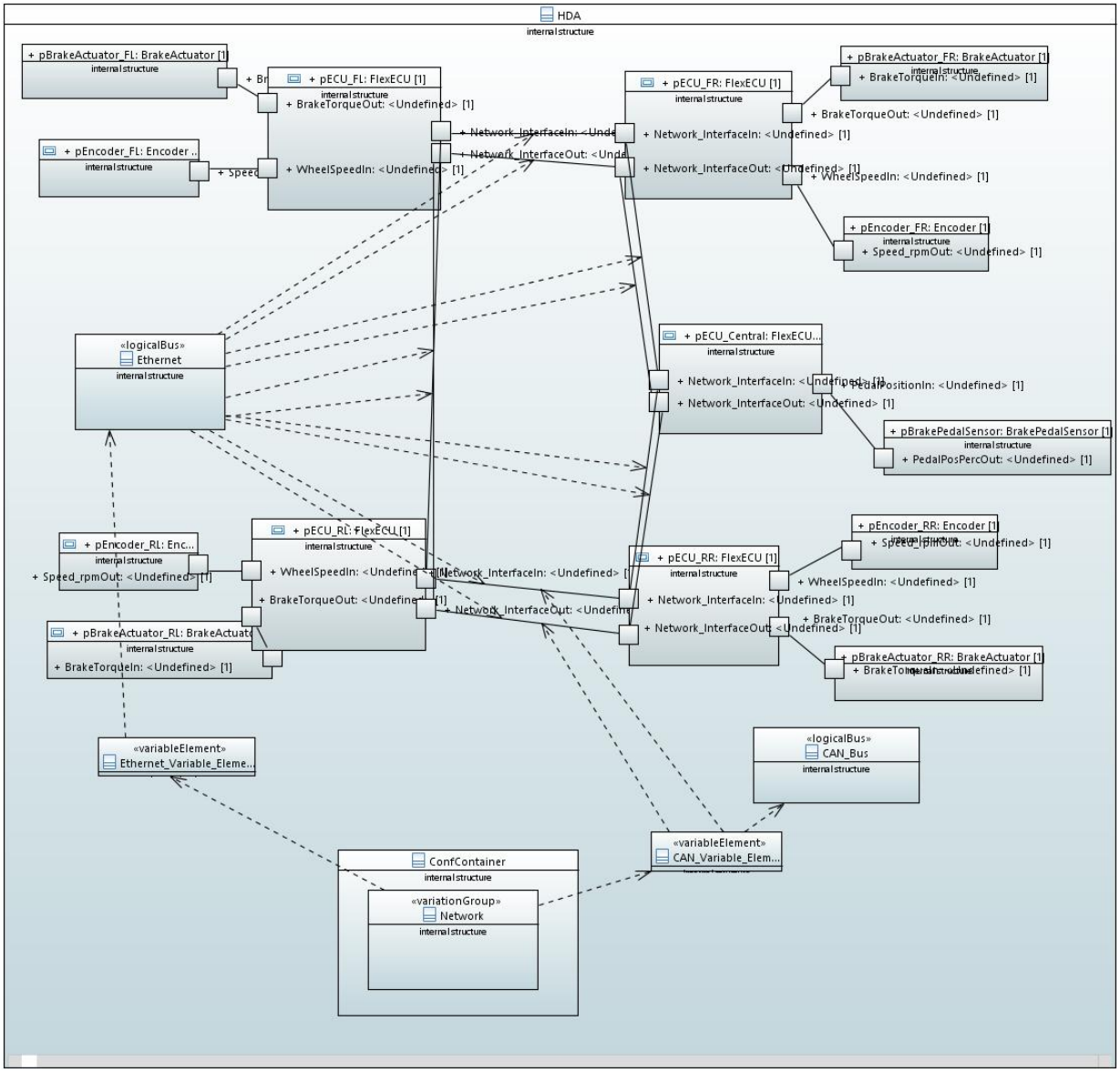

Figure 6: HDA in Papyrus

- Allocations

  Figure 7 shows FDA-HDA Allocation diagram in Papyrus. This diagram represents the FDA as a DesignFunctionType and the HDA as a DesignFunctionPrototype. Through the allocation rows, the allocations between DesignFunctionPrototypes and HardwareFunctionPrototypes are represented.

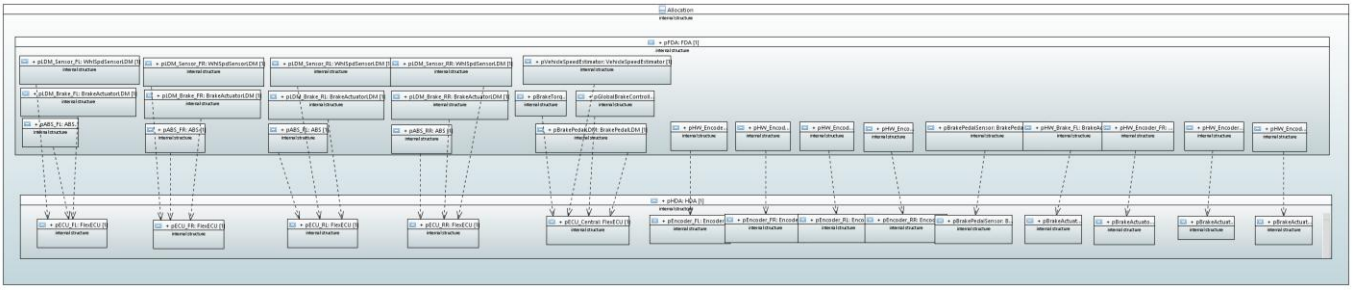

Figure 7: Allocations in Papyrus

5.4 BBW in EATOP

- Functional Design Architecture.

  As EATOP does not provide a graphical representation of its models, *Figure 8* shows the FDA Explorer tree that contains some of the FDA children.

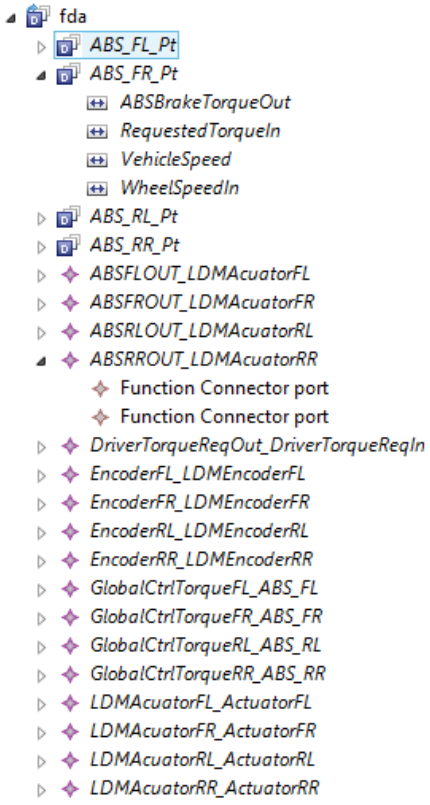

Figure 8: EATOP's FDA tree

- Hardware Design Architecture

  Figure 9 shows the FDA Explorer tree that contains some of the FDA children.

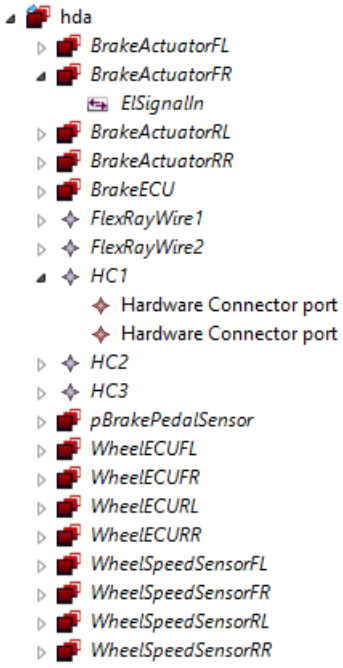

Figure 9: EATOP's HDA tree

- Allocations

  Figure 10 shows the FunctionAllocations defines in the BBW use case.

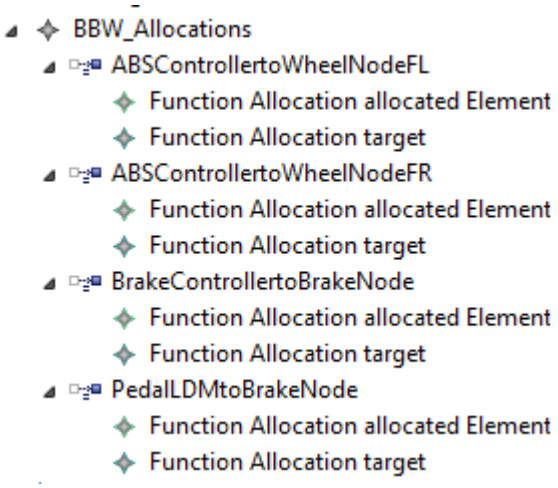

Figure 10: FunctionAllocations in EATOP

Figure 11 shows the allocated element and the target of each FunctionAllocation contained in *Figure 10*.

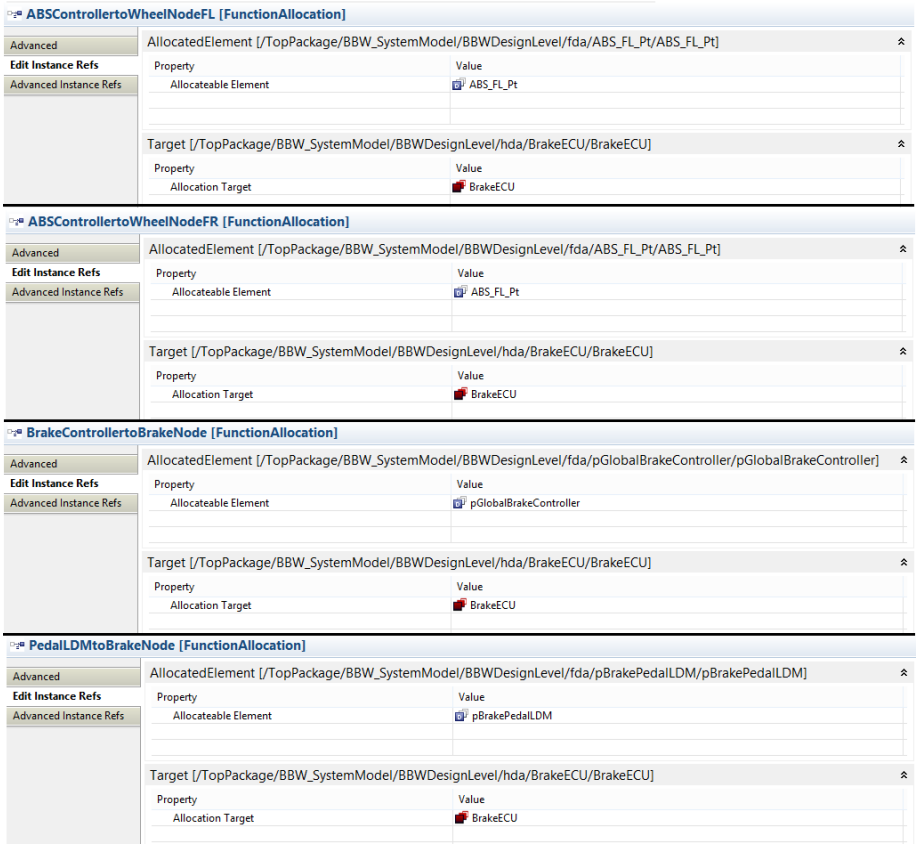

Figure 11: Target and allocated element

5.5 BBW in MetaEdit+

- Functional Design Architecture.

  Figure 12 shows FDA diagram in MetaEdit+. We can appreciate that this diagram differs from the Papyrus one in the following: this diagram does not provide input wheel elements such as sensors or encoders as they will be provided by the HDA. The same happen with speed vehicle calculation and actuators.

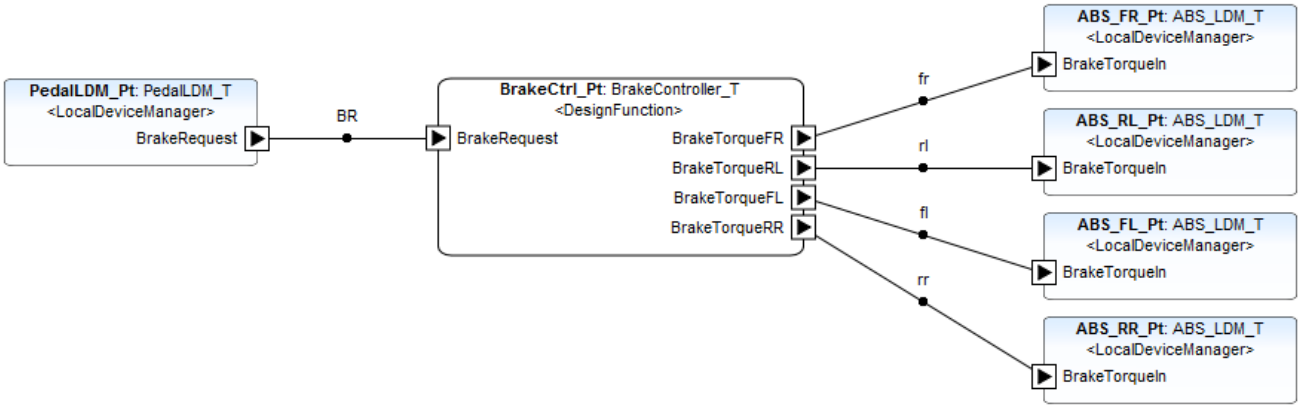

Figure 12: FDA in MetaEdit+

- Hardware Design Architecture

  Figure 13 shows the HDA diagram in MetaEdit+. This diagram represents braking level provided by sensors and a lock button. It calculates the amount of brake pressure, taking into account vehicle speed calculated and locks operation.

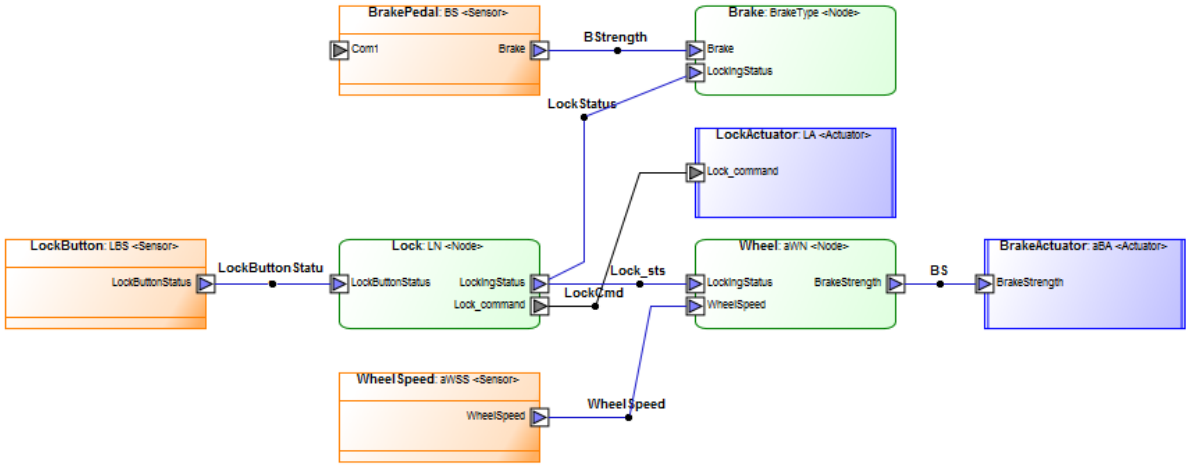

Figure 13: HDA in MetaEdit+

- Allocations

  Figure 14 represents Allocation table in MetaEdit+, where the allocations between BBW elements are shown.

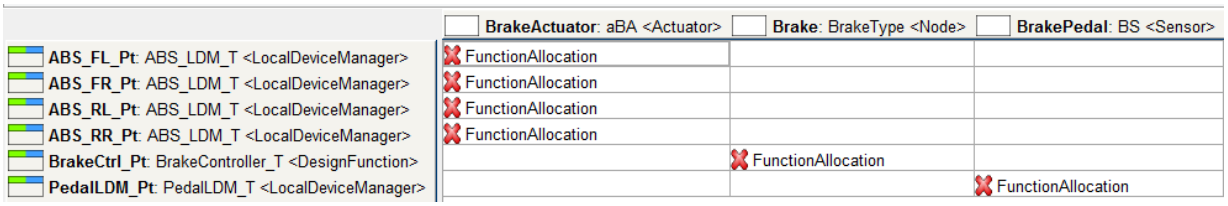

Figure 14: Allocations table in MetaEdit+

Allocations can also be shown in MetaEdit+ on HDA diagrams.

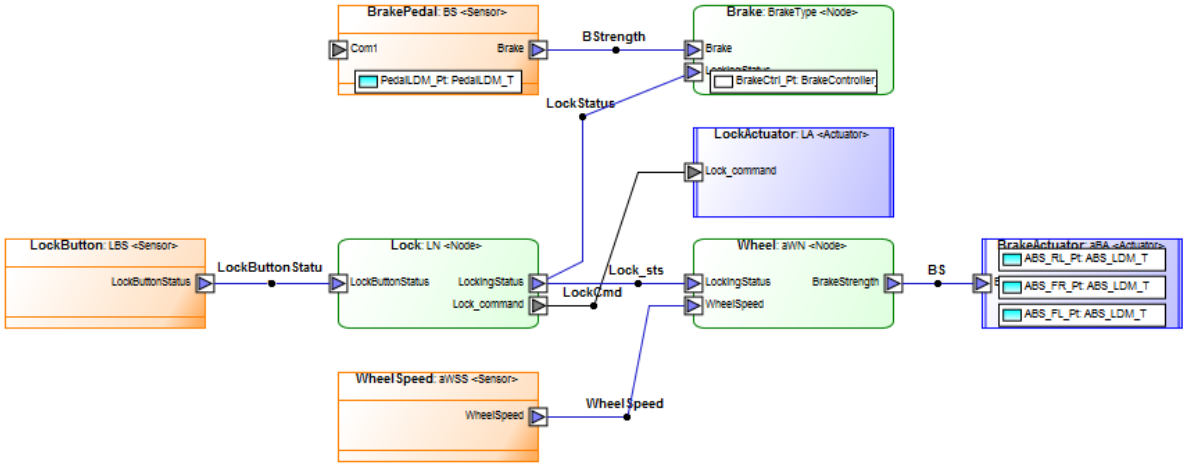

Figure 15: HDA with allocations diagram in MetaEdit+

6. Conclusion and future work

In this project, a comparison between three EAST-ADL modeling tools (Papyrus, EATOP and MetaEdit+) has been presented in order to provide developers a guide that offers some tool information to facilitate their choice for developing their models. For achieving that, firstly, a basic knowledge of both, the ADL language used and the tools to compare, has been provided. After that, the comparison criteria has been defined, and subsequently applied. Finally, a particular use case has been shown, which visually reflects the features already mentioned of each tool.

As limitations, each tool has its own constraints that hinder the learning process such as, for instance, the application of generic constraints in Papyrus, the lack of graphical interface in EATOP or the fact that MetaEdit+ is not an open source tool. However, these circumstances have been the key to make a good distinction between the three tools.

As future work, other modeling tools that support EAST-ADL have been developed or are in development process, such as SystemWeaver[24], MagicDraw[25] or Mentor Graphics VSA[26], that can be taken into account when comparing these kind of tools on future works.

References

[1] Blom, Lönn, et al. "EAST-ADL - An Architecture Description Language for Automotive Software-Intensive Systems." White Paper. Version M2.1.10.
[2] EAST-ADL Introduction. EAST-ADL tooling. MAENAD.
[3] EAST-ADL Introduction. EAST-ADL overview. MAENAD.
[4] EAST-ADL Domain Model Specification. Version V2.1.12.
[5] Papyrus Proposal: Website (http://wiki.eclipse.org/MDT/Papyrus-Proposal, last visit 2015-05-10).
[6] Lanusse, Agnes, et al. "Papyrus UML: an open source toolset for MDA."Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications, 2009.
[7] EATOP proposal website (http://eclipse.org/proposals/modeling.eatop/, last visit 2015-05-12).
[8] MetaEdit+ EAST-ADL Tutorial. Version 5.1. MetaCase Document No. EAT-5.1. 8th edition April 2015.
[9] MetaCase manual: Website (https://www.metacase.com/support/45/manuals/meplus/Mp.html, last visit 2015-05-11).
[10] Papyrus Eclipse website (http://www.eclipse.org/papyrus/, last visit 2015-05-17).
[11] EATOP Eclipse website (http://www.eclipse.org/eatop/download.php, last visit 2015-05-12).
[12] MetaCase webpage (http://www.metacase.com, last visit 2015-06-03).
[13] Papyrus documentation (http://www.eclipse.org/papyrus/usersCorner/usersCornerIndex.php, last visit 2015-06-03).
[14] Wiki EATOP website (http://wiki.eclipse.org/EATOP, last visit 2015-06-03).
[15] MetaCase manual: Website (https://www.metacase.com/support/45/manuals/meplus/Mp.html, last visit 2015-05-11).
[16] MAENAD website presentations (http://www.maenad.eu/presentations.html, last visit 2015-06-03).
[17] Eclipse website downloads (http://www.eclipse.org/downloads/, last visit 2015-05-17).
[18] Papyrus download website (http://www.eclipse.org/papyrus/downloads/, last visit 2015-06-03).
[19] EATOP demonstrator download website (http://www.eclipse.org/eatop/download.php, last visit 2015-06-03).
[20] Artop Initiative website (www.artop.org, last visit 2015-05-17).
[21] Brake-by-wire webpage (http://www.brakebywire.com/, last visit 2015-06-15)
[22] Brake-by-wire description from Wikipedia, the free encyclopedia (last visit 2015-06-10).
[23] Qureshi, Tahir Naseer, et al. "From EAST-ADL to AUTOSAR Architecture." (2011).
[24] Systemite webpage (http://systemite.se/news-events?page=1, last visit 2015-06-15).
[25] NoMagic webpage (http://www.nomagic.com/products/magicdraw.html, last visit 2015-06-15)
[26] Mentor graphics webpage (http://www.mentor.com/products/vnd/, last visit 2015-06-15)